

## FIGURE 1

TGGCCTCCCCAGCTGCCAGGCACAAGGCTGAGCGGGAGGAAGCGAGAGGCATCTA  
AGCAGGCAGTGTTGCCCTCACCCAAAGTGA**CATGAGAGGTGCCACCGAGTC**  
AACATGCTCCTCCTAGTA**ACTGTGCTGACTGTGCTGTGATCACAGGGGCTGTGA**  
GC~~GGG~~GATGTCCAGTGTGGGGCAGGCACCTGCTGCCATCAGCCTGTGGCTCGAGG  
GCTGCGGA**TGTGCA**CCCCGCTGGGGAGGCAGGAGTGC**CCACCCGGCAGCC**  
ACAAGGTCCCCTCTCAGGAAACGCAAGCACACAC**TGTGCTGCCATGGACTGAA**ACA  
TGCTGTGCTCCAGGTTCCGGACGGCAGGTACCGCTGCTCCATGGACTGAA**ACA**  
TCAATT**TAGGCGCITGCC**CTGGTCTCAGGATA**ACCCACCATCCTTCTGAGCACAG**  
CCTGGATTTTATTTCTGCCATGAAACCCAGCTCCATGACTCTCCAGTCCCTACAC  
TGACTACCCCTGATCTCTTGTCTAGTACGCACATATGCACACAGGCAGACATACCT  
CCCACATGACATGGTCCCCAGGCTGGCTGAGGATGTACAGC**ITGAGGCTGTGGT**  
GTGAAAGGTGGCCAGCCTGGTCTCTCCCTGCTCAGGCTGCCAGAGAGGTGGTAAA  
TGGCAGAAAGGACATTCCCCCTCCCCACATGTATCCCTCGGTCTGAATTAGACATTCTGGCACAGGCTC  
TGCCCCCTCTCCCCACATGTATCCCTCGGTCTGAATTAGACATTCTGGCACAGGCTC  
TTGGGTGCATTGCTCAGAGTCCCAGGTCCCTGGCCTGACCTCAGGCCCTCACGTGA  
GGTCTGTGAGGACCAATTGTGGGTAGTTCATCTCCCTCGATTGGTTACTCCTTAG  
TTTCAGACCACAGACTCAAGATTGGCTTCCCAGAGGGCAGCAGACAGTCACCCCA  
AGGCAGGTGTAGGGAGCCCAGGGAGGCCAATCAGCCCCCTGAAGACTCTGGTCCA  
GTCAGCCTGTGGCTTGTGGCCTGTGACCTGTGACCTTCTGCCAGAATTGTATGCCTC  
TGAGGCCCTTACACACTTACCA**GTTAACCA**CTGAAGCCCCAATTCCACA  
GCTTTCCATTAAAATGCAAATGGTGGTTCAATCTAATCTGATATTGACATATTA  
GAAGGCAATTAGGGTGTTCCTAAACA**ACTCCTTCCAAGGATCAGGCCCTGAGAGC**  
AGGTTGGT**GACTTGTAGGAGGGCAGTCC**AGATTGGGTGGAGCAAGGG  
ACAGGGAGCAGGGCAGGGGCTGAAAGGGGACTGATT**CAGACCAGGGAGGCAACT**  
ACACACCAACATGCTGGCTT**AGAATAAAAGCACCAACTGAAAAAA**



## **FIGURE 2**

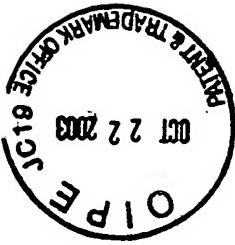
MRGATRVSIMLLVTVDCAVITGACERDVQCGAGTCCAISLWLRGLRMCTPLGREGE  
C  
HPGSHKVPFFRKHKHTCPCLPNLLCSRFPDGRYRCSDLKNINF

Important features:

Signal peptide:  
1-19

N-myristoylation sites:

33  
35  
46



## FIGURE 3A

PRO	XXXXXXXXXXXXXXXXXXXX	(Length = 15 amino acids)
Comparison Protein	XXXXXXYYYYYYYY	(Length = 12 amino acids)

% amino acid sequence identity =

(the number of identically matching amino acid residues between the two polypeptide sequences as determined by ALIGN-2) divided by (the total number of amino acid residues of the PRO polypeptide) =

5 divided by 15 = 33.3%



## **FIGURE 3B**

PRO	XXXXXXXXXX	(Length = 10 amino acids)
Comparison Protein	XXXXXYYYYYYYZZYZ	(Length = 15 amino acids)

% amino acid sequence identity =

(the number of identically matching amino acid residues between the two polypeptide sequences as determined by ALIGN-2) divided by (the total number of amino acid residues of the PRO polypeptide) =

5 divided by 10 = 50%



### **FIGURE 3C**

PRO-DNA

NNNNNNNNNNNNNNN

(Length = 14 nucleotides)

Comparison DNA

NNNNNNNNNNNNNNN

(Length = 16 nucleotides)

% nucleic acid sequence identity =

(the number of identically matching nucleotides between the two nucleic acid sequences as determined by ALIGN-2) divided by (the total number of nucleotides of the PRO-DNA nucleic acid sequence) =

6 divided by 14 = 42.9%



## **FIGURE 3D**

PRO-DNA

NNNNNNNNNNNN

(Length = 12 nucleotides)

Comparison DNA

NNNNLLLVVV

(Length = 9 nucleotides)

% nucleic acid sequence identity =

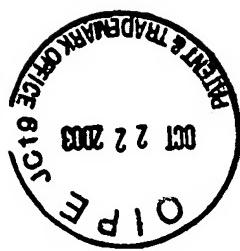
(the number of identically matching nucleotides between the two nucleic acid sequences as determined by ALIGN-2) divided by (the total number of nucleotides of the PRO-DNA nucleic acid sequence) =

4 divided by 12 = 33.3%

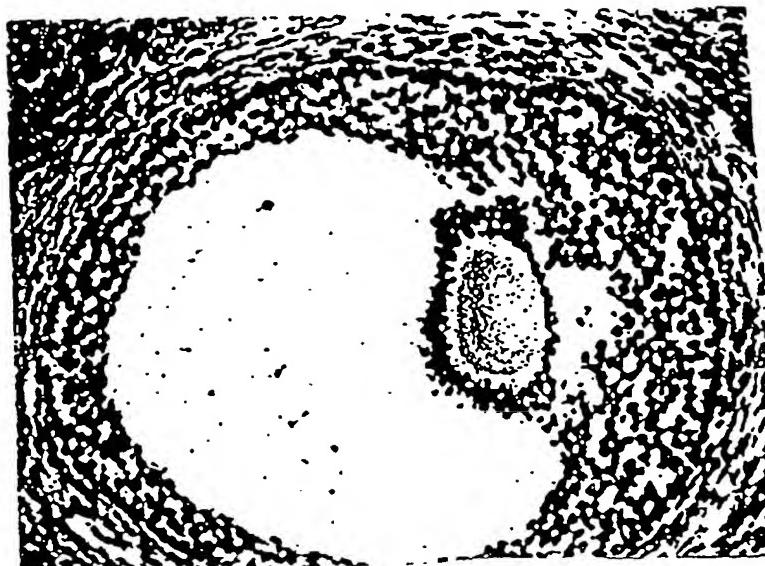


## **FIGURE 4**

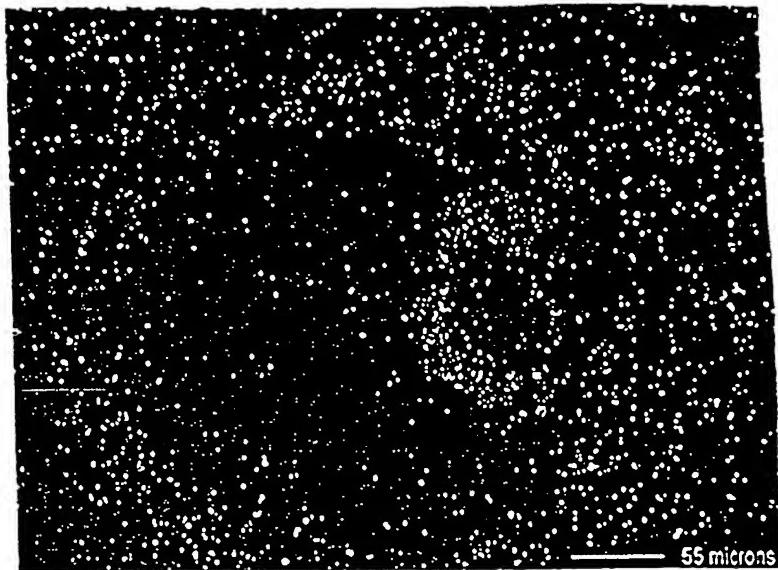
TGGCTCCCCAGCTTGCCAGGCACAAGGCTGAGCTGGAGGAAGCGAGANGCATCTAA  
GCAG  
GCAGTGTTTGCCTTCACCCCAAGTGACCATGAGAGGTGCCACGCGAGTCTCAATCA  
TGC  
TCCCTCCTAGTAACTGTGTCTGACTGTGCTGTGATCACAGGGGCCTGTGAGCGGGATG  
TCC  
AGTGTGGGCAGGCACCTGCTGTGCCATCAGCCTGTGGCTCGAGGGCTGCGGATGT  
GCA  
CCCCGCTGGGCGGGAAAGGCGAGGAGTGCCACCCGGCAGCCACAAGGTCCCCTTC  
TTCA  
GGAAACGCAAGCACACACCTGCTTGTGCCAACCTGCTGTGCTCCAGTTCCGGA  
CGG  
CACTACGCTGCTCA



**FIGURE 5A**

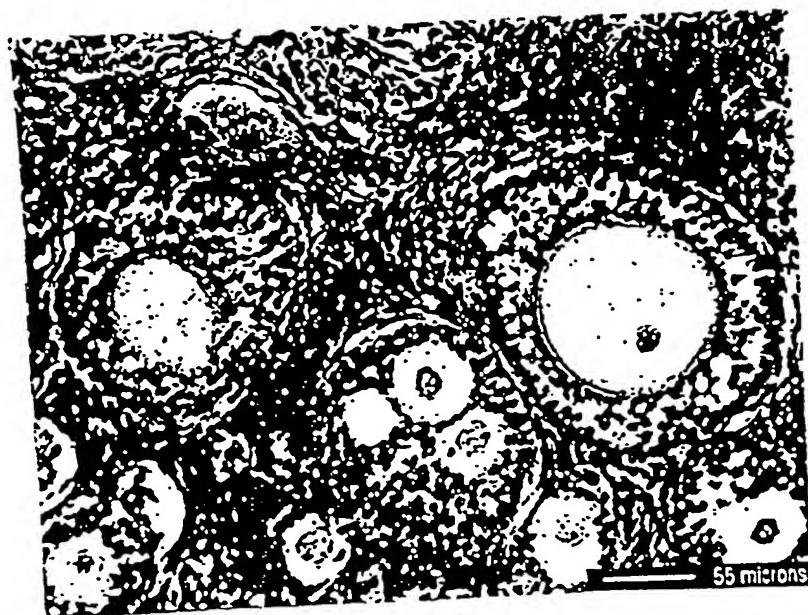


**FIGURE 5B**

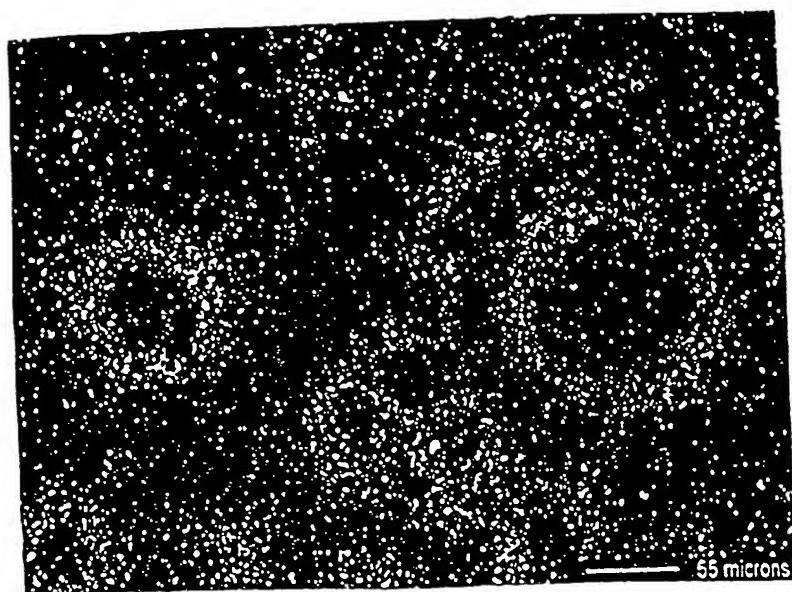


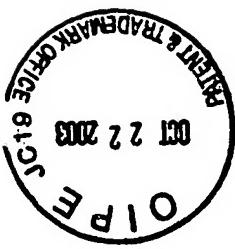


**FIGURE 5C**

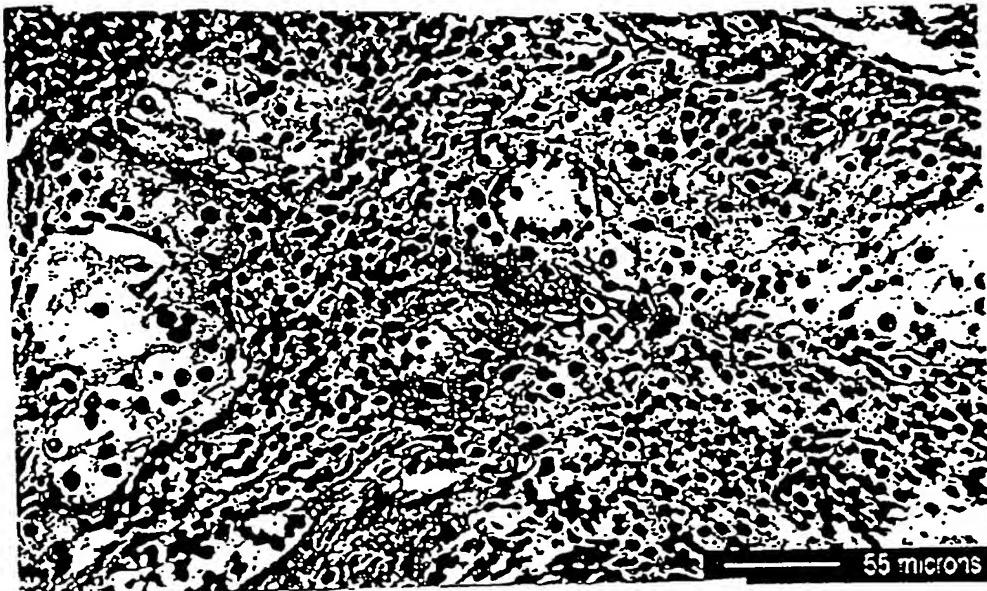


**FIGURE 5D**

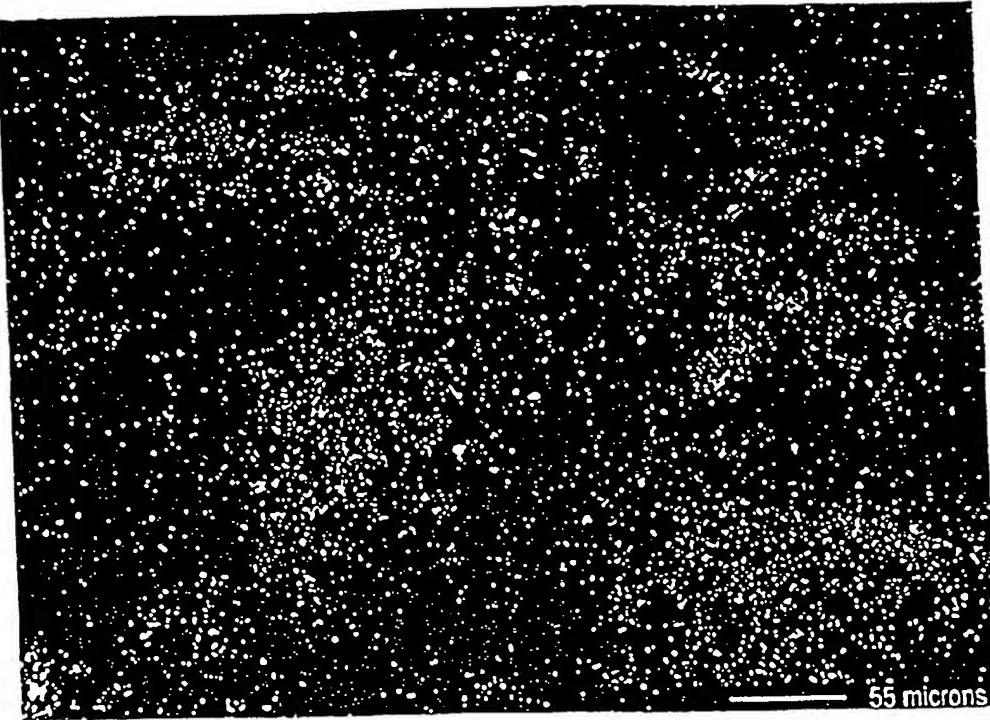




**FIGURE 5E**

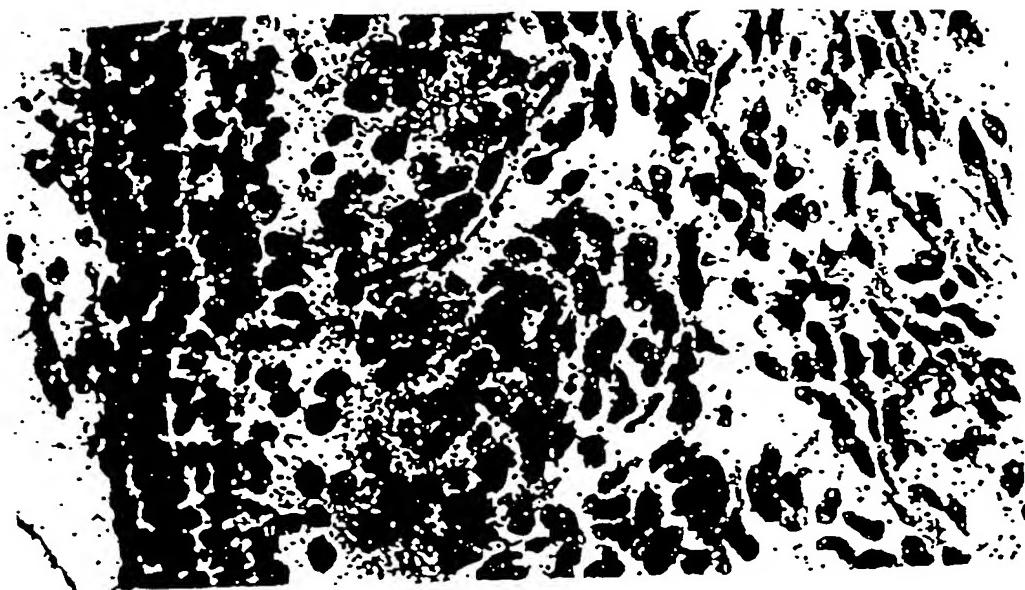


**FIGURE 5F**



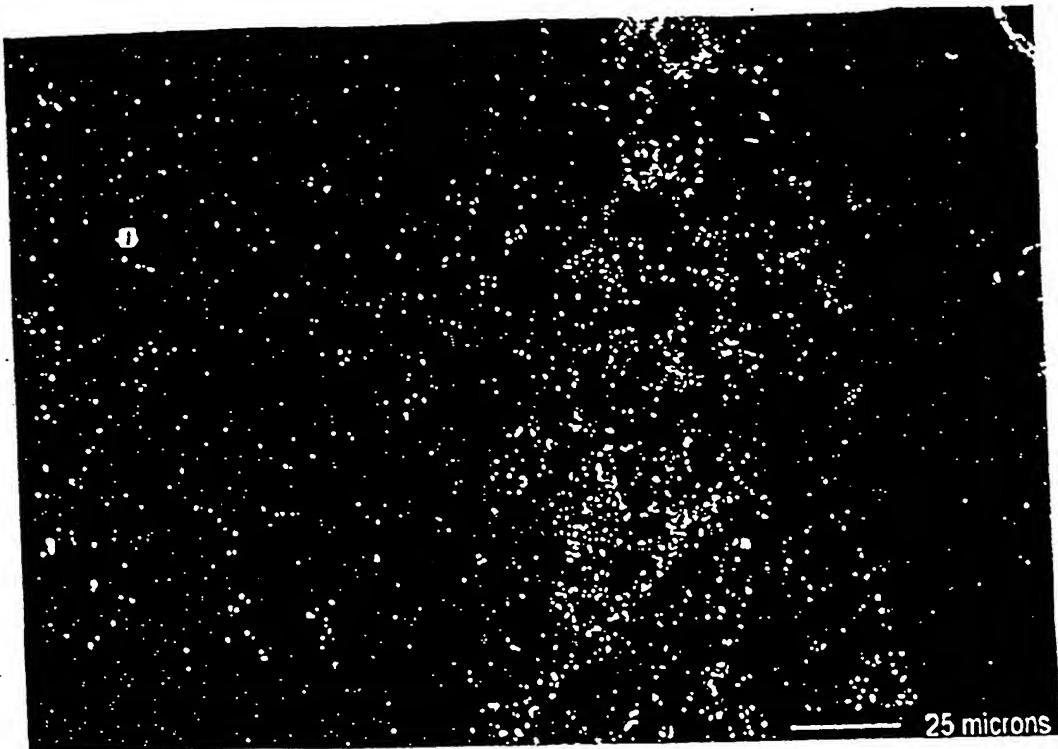


## FIGURE 6A





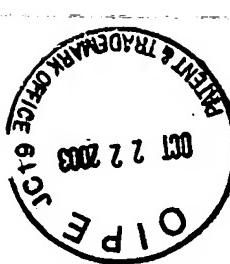
## FIGURE 6B



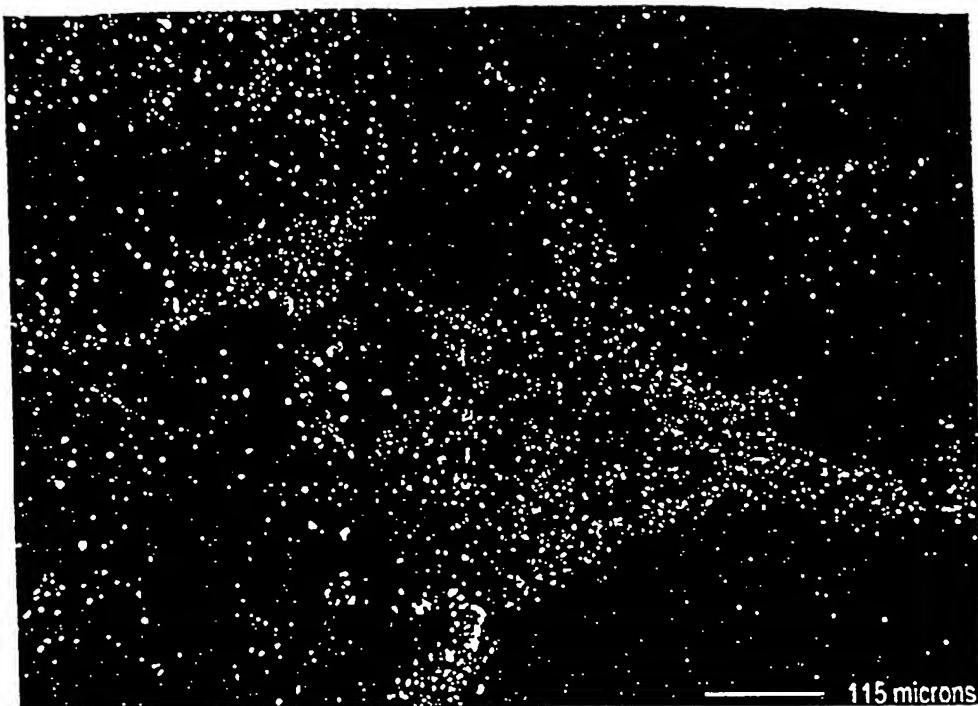


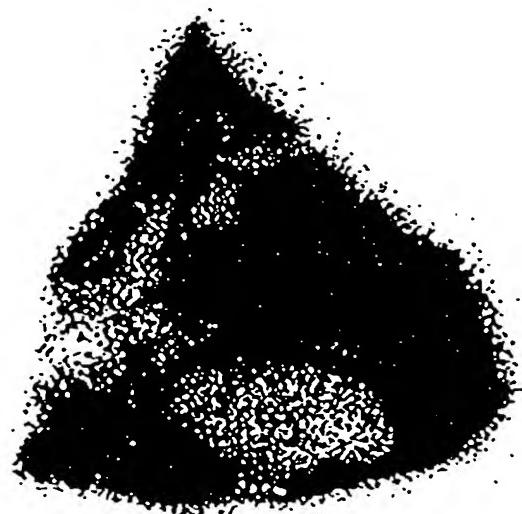
## FIGURE 6C





## FIGURE 6D





**FIGURE 7A**



**FIGURE 7B**



FIGURE 8

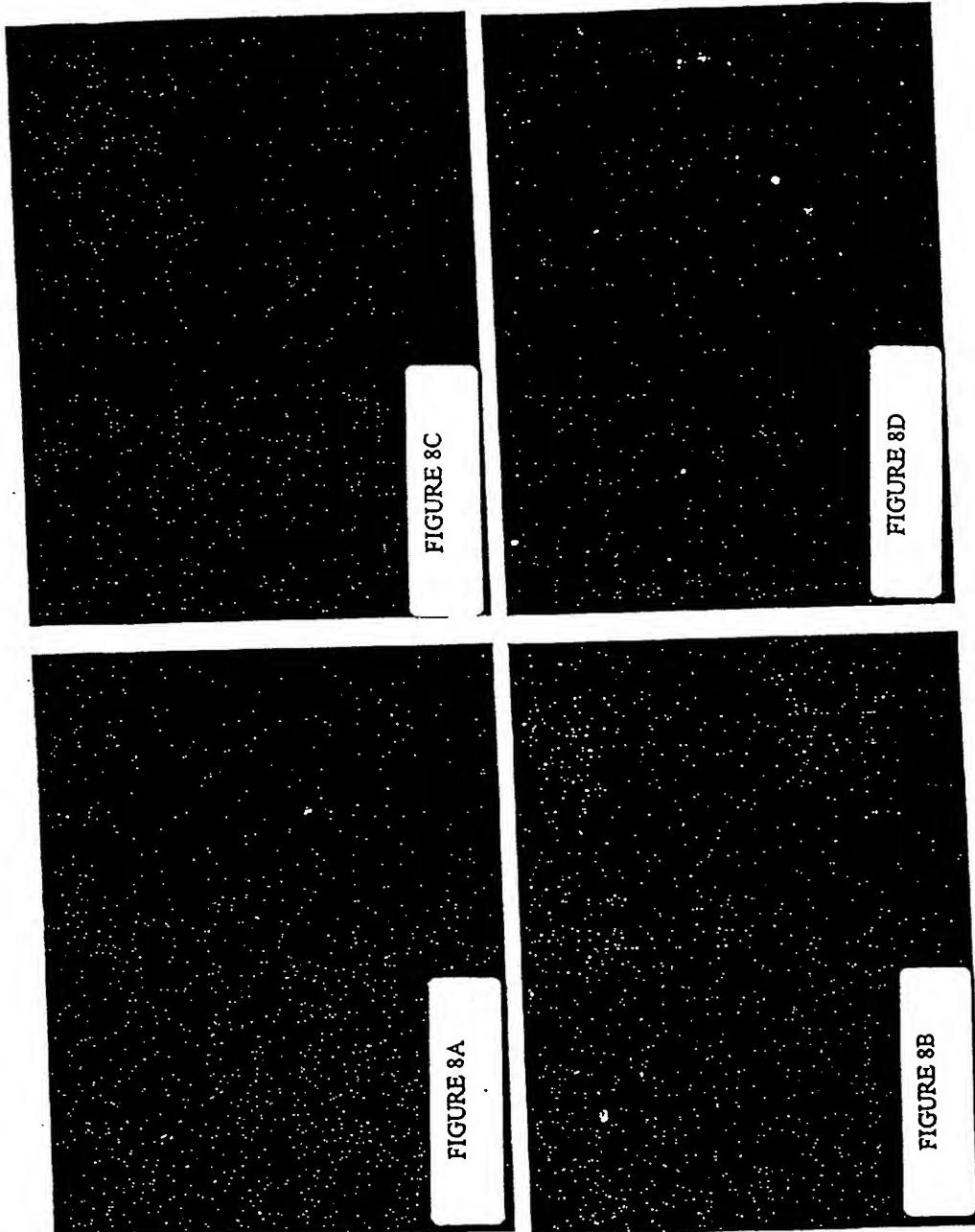




FIGURE 9

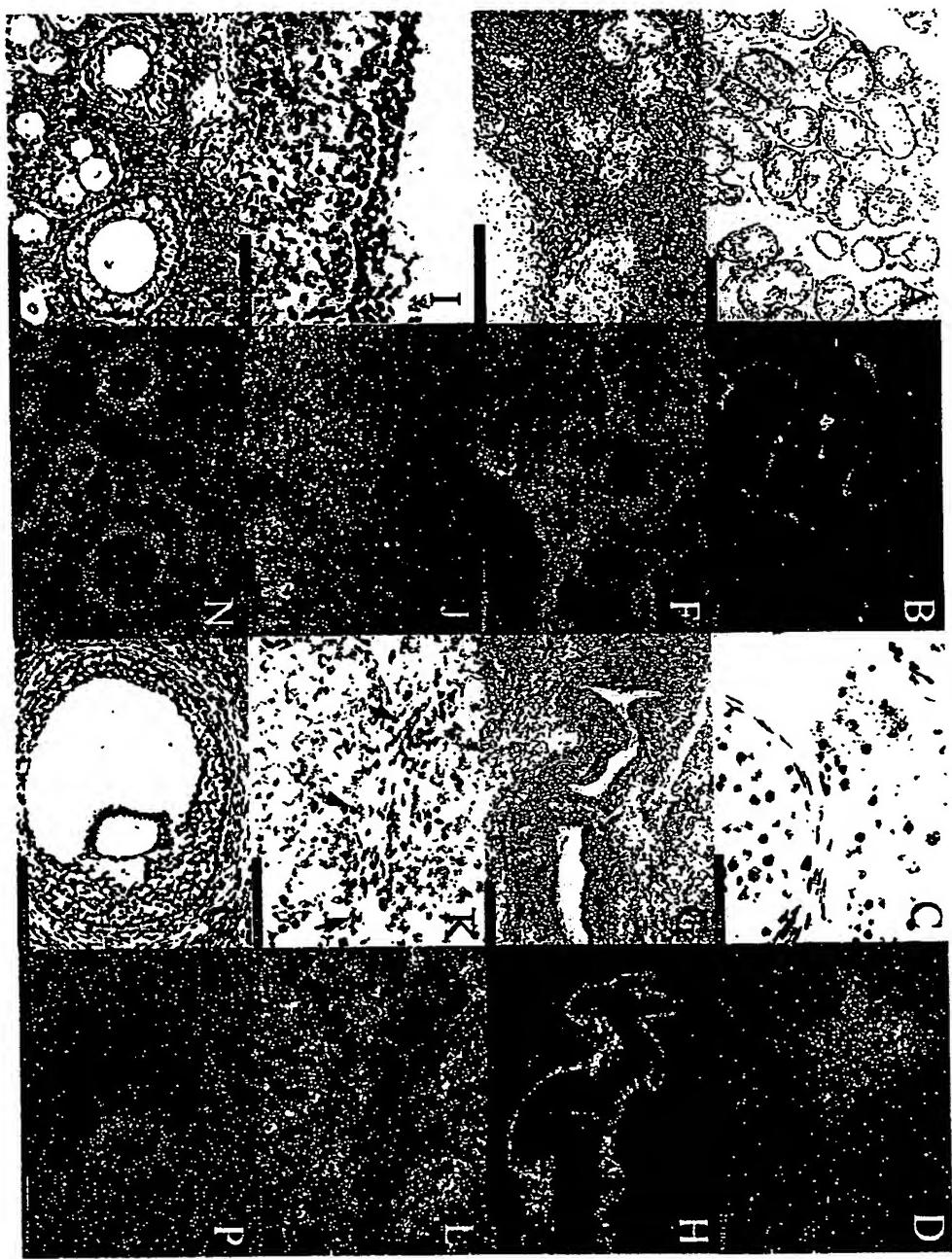
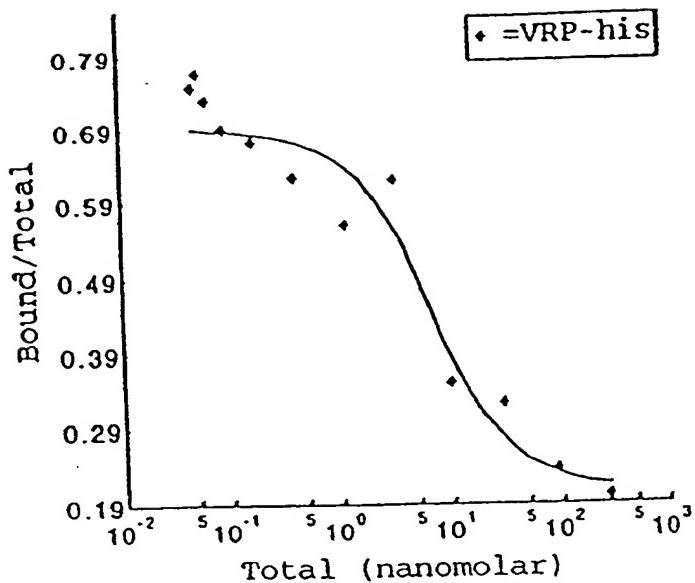


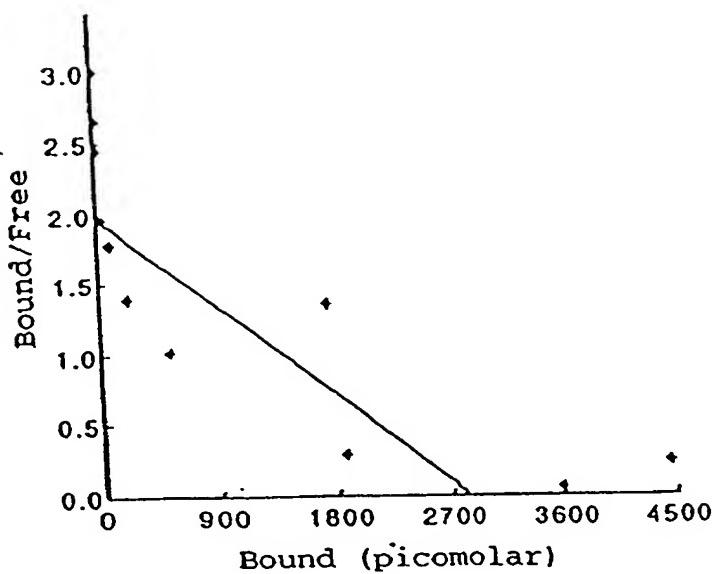


FIGURE 10 A



$$K_d = 1.43 \pm 0.4 \text{ nM}$$

FIGURE 10 B



**FIGURE 10 C**

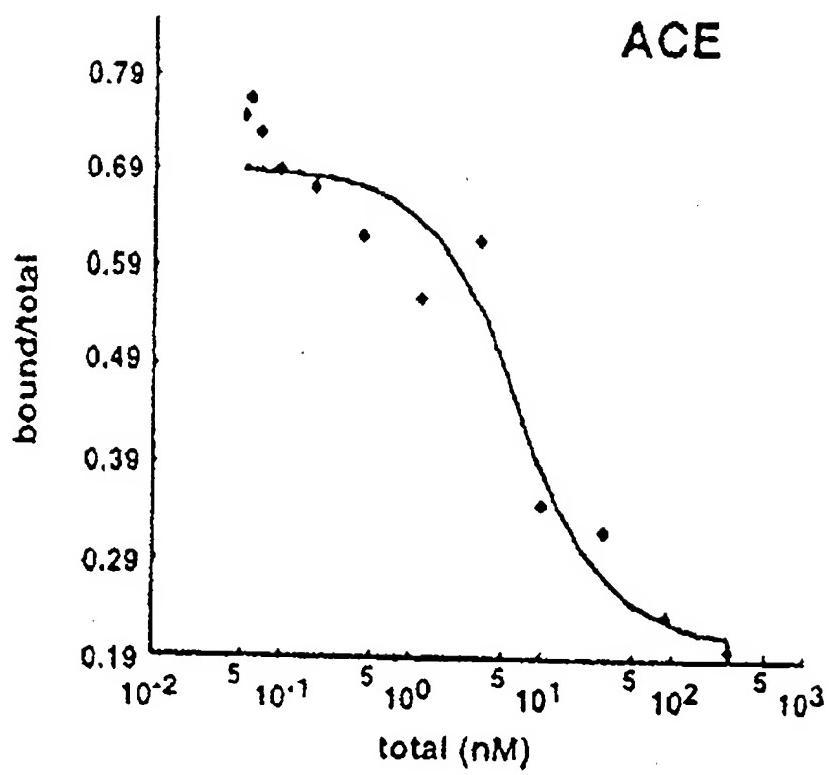
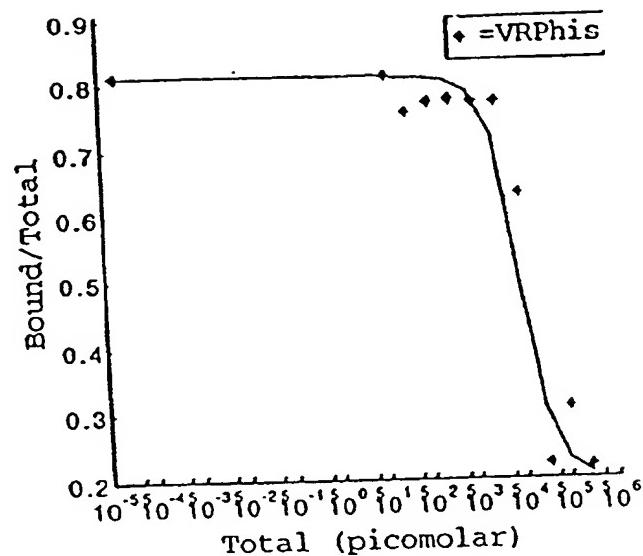


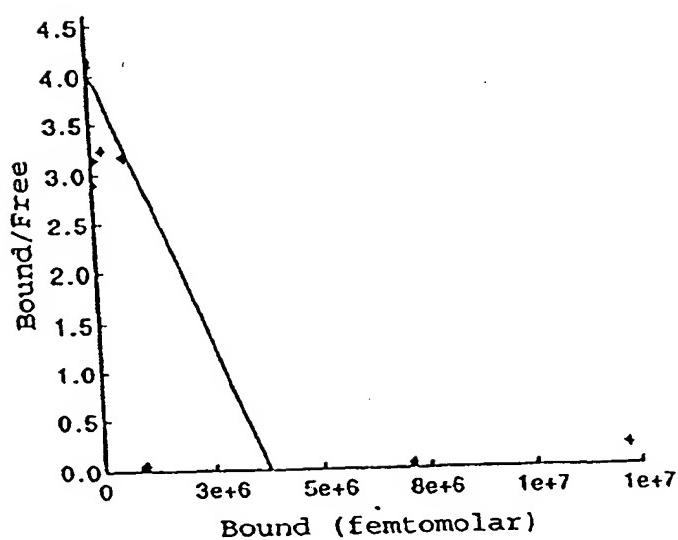


FIGURE 11 A



$$K_d = 0.95 \pm 0.6 \text{ nM}$$

FIGURE 11 B





**FIGURE 11 C**

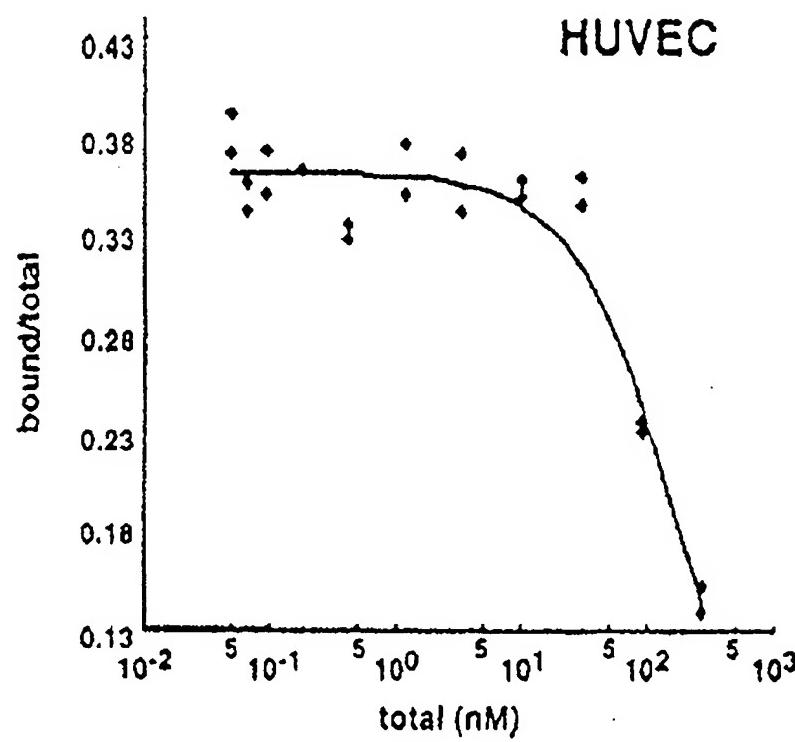




FIGURE 12 A

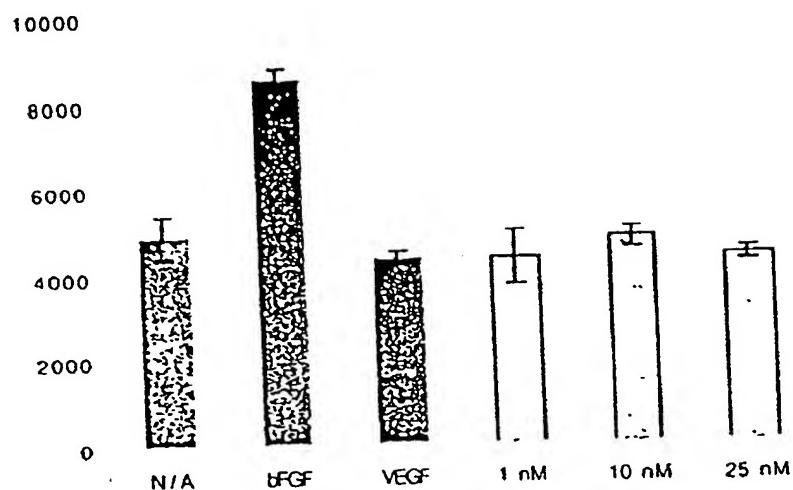




FIGURE 12 B

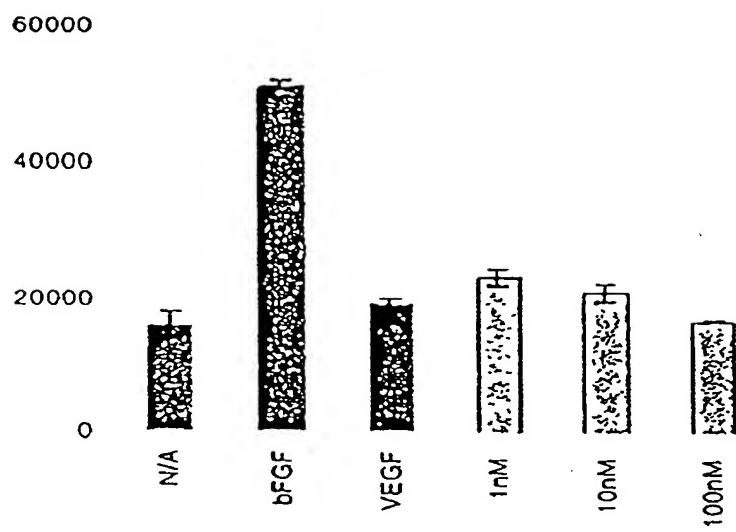




FIGURE 12 C

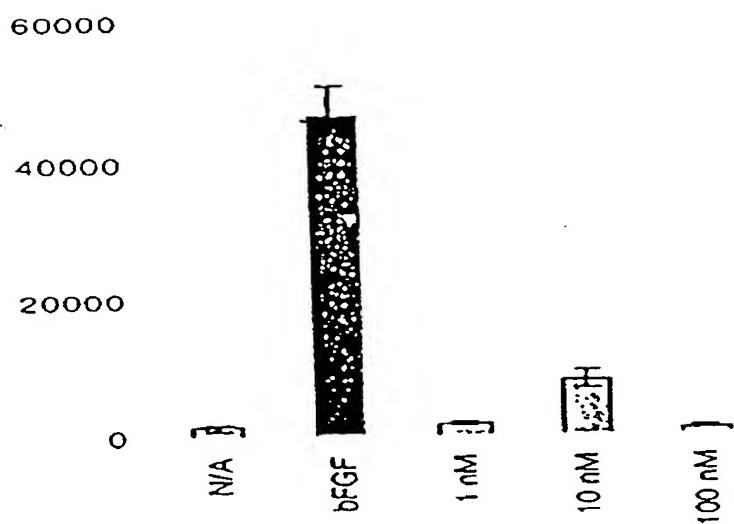
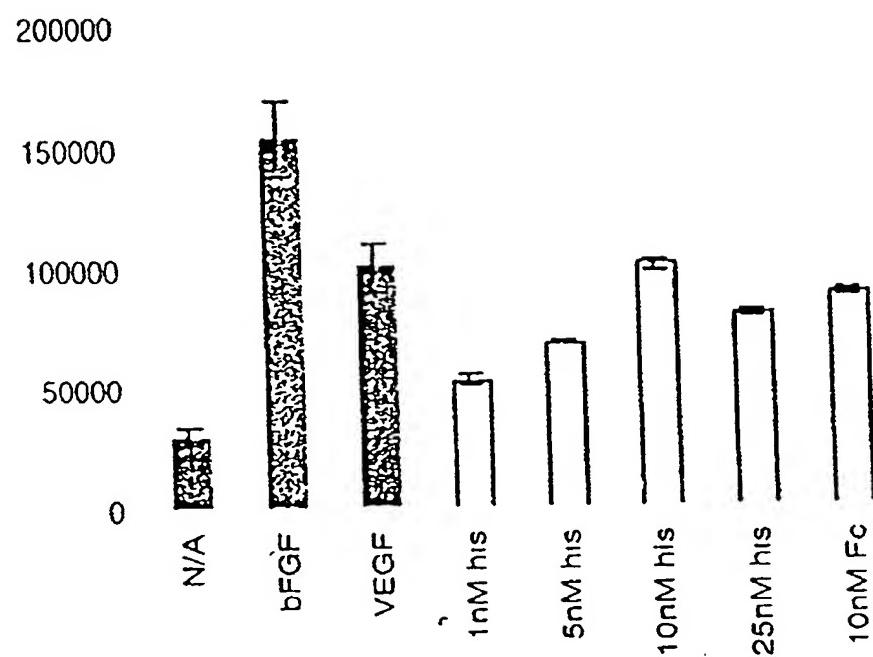


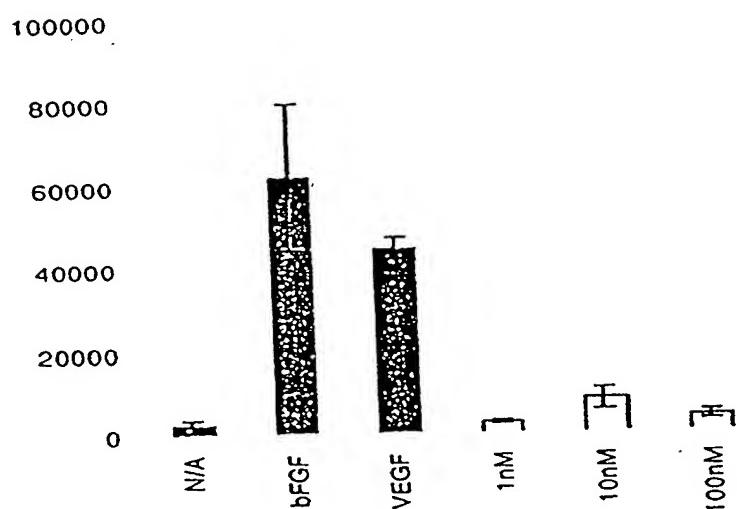


FIGURE 12 D



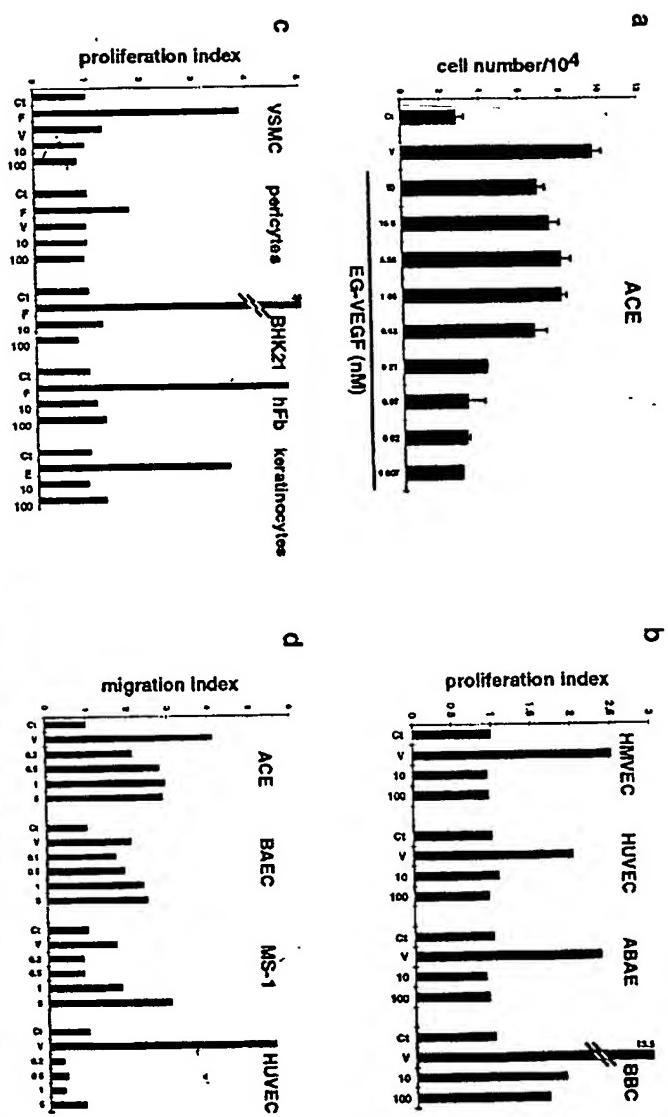
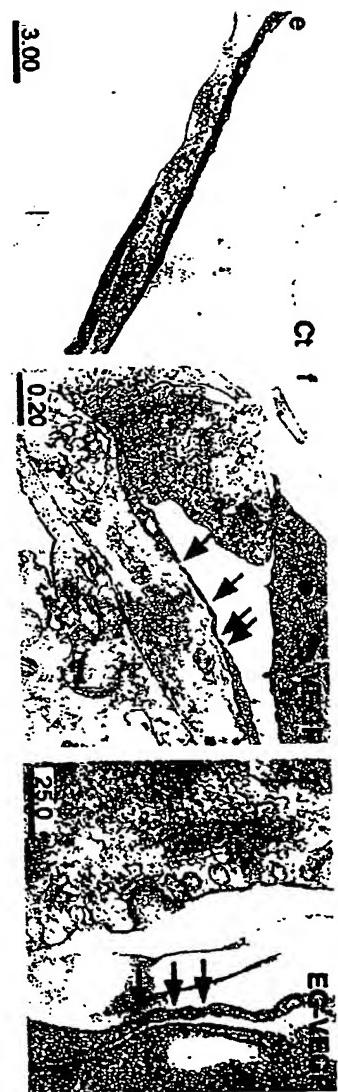
PRINTED MATERIAL  
OCT 22 2003  
OIPM

FIGURE 12 E



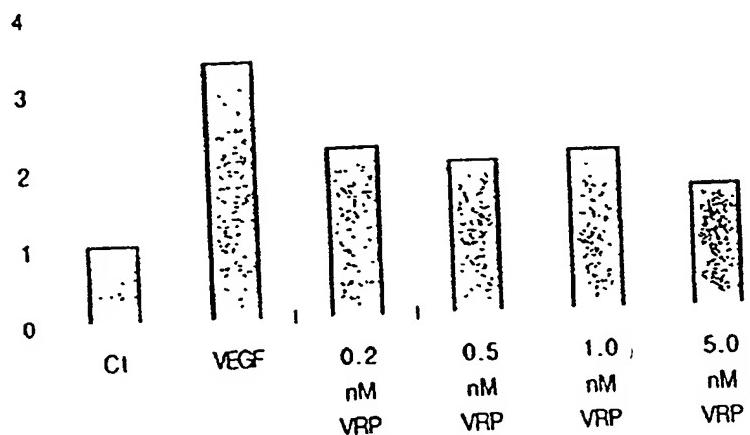
OPIU JC19 31/12/2003  
DET 22 2003  
REGISTERED TRADEMARK  
GPT

FIGURE 13





**FIGURE 14 A**



**FIGURE 14 B**

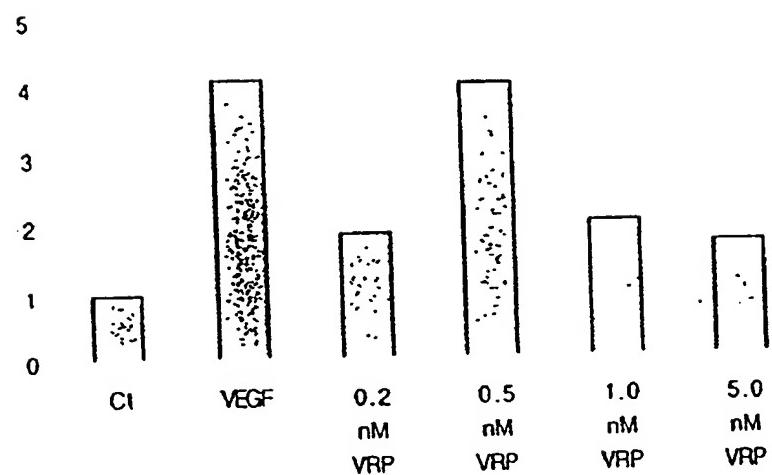
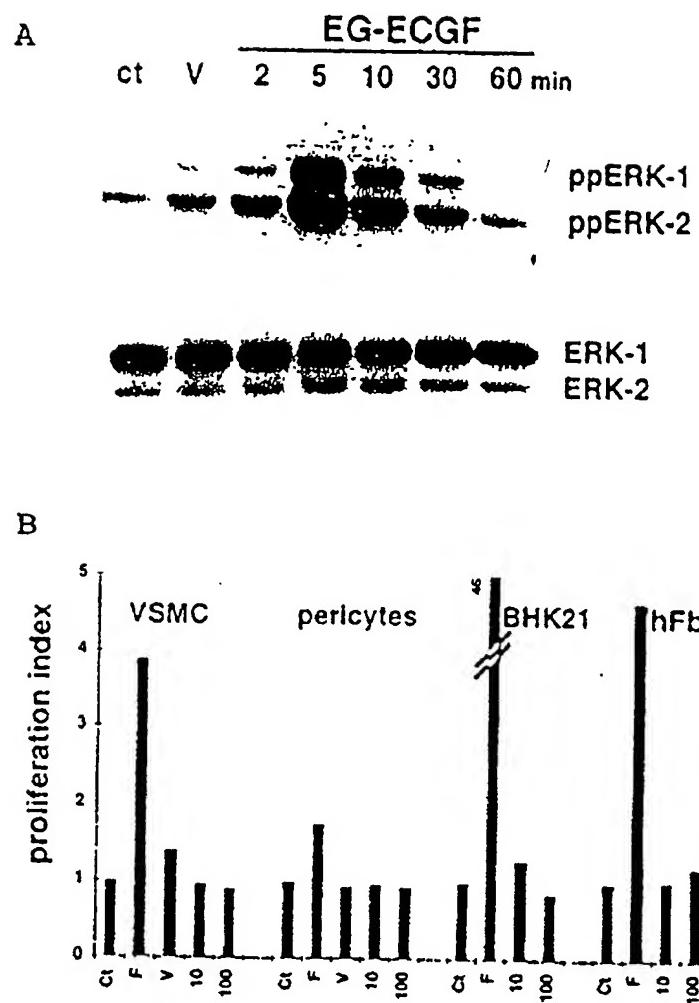
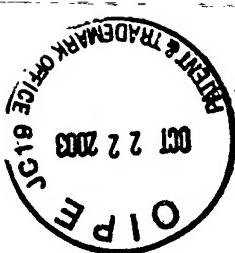




FIGURE 15





**FIGURE 16 A-C**

**a**

```

TGGCCTCCCCAGCTGCCAGGCACAAGGCTGAGCGGGAGGAAGCGAGAGG 50
CATCTAACGCAGGCAGTGTTTGCCTTCACCCCAAGTGACCATGAGAGGTG
M R G
CCACCGCAGACTCAATCATGCTCCTCTAGTAACCTGTGCTGACTGTGCT 200
A T R V S I M L L L V T V S D C A
GTGATCACAGGGGCCTGTGAGCGGGATGTCAGTGAGGGCAGGCACCTG
V I T G A C E R D V Q C G A G T C
CTGTGCATCAGCCTGTGGCTTCGAGGGCTGCGGATGTGACCCCGCTGG 400
C A I S L W L R G L R M C T P L
GCCGGAAAGCCGAGGACTGCCACCCGGCAGCCACAAGTCCCCTCTTC
G R E G E E C H P G S H K V P F F
AGGAAACGCAAGCACACCTGTCCCTGCCCCAACCTGCTGTGCTC
R K R K H H T C P C L P N L L C S
CAGGTTCCCGAGCCGAGGTACCGCTGCTCATGGACTTGAAGAACATCA
R F P D G R Y R C S M D L K N I
ATTTTAGGCCTTGCCCTGCTCAGGATACCCACCATCCTTTCTGAG
N F *
CACAGCCTGGATTTTATTCTGCCATGAAACCCAGCTCCCATGACTCTC 600
CCAGTCCCTACACTGACTACCCCTGATCTCTCTGTCTAGTACCCACATAT
GCACACAGGCAGACATACCTCCCATCATGACATGGTCCCCAGGCTGGCCT
GAGGATGTCACAGCTTGAGGCTGTGGGTGAAAGGTGGCCAGCTGGTTC
CTTCCCTGCTCAGGCTGAGGAGGGTGGTAAATGGCAGAAAGGACATT 800
CCCCCTCCCTCCCCACGTGACCTGCTCTTCTGGGGCCCTGCCCCTC
TCCCCACATGTATCCCTCGGTCTGAATTAGACATTCTGGCACAGGCTC
TTGGGTGCAATTGCTCAGGCTCCAGGTCTGGCTGACCCCTAGGCCCTT
CACGTGAGGTCTGTGAGGACCAATTGGTGGTAGTTCATCTTCCCTCGAT
TGGTTAACCTCTTAGTTTCAAGACCACAGACTCAAGATTGGCTCTTCCAG 1000
AGGGCAGCAGACAGTCACCCCAAGGAGGGTGTAGGGAGGCCAGGAGGCC
AATCAGCCCCCTGAAGACTCTGGTCCCAGTCAGCCTGGCTGTGGCT
GTGACCTGTGACCTCTGCCAGAATTGTCATGCCCTCTGAGGCCCTTCTT
ACCACACTTACCAAGTTAACACTGAAGCCCCAAATCCCCACAGCTTTTC
CATTTAAATGCAAATGGTGGTCAATCTAAATCTGATAATTGACATATT 1200
AGAAGGCAATTAGGGTGTCTTAAACAACCTCTTCCAAGGATCAGCC
CTGAGACCAGGTGGTAGTTGAGGAGGGCAGTCTGTGCCAGATTGG
GGTGGGAGCAAGGGACAGGAGCAGGGCAGGGCTGAAAGGGCACTGAT
TCAGACCAGGGAGGCAACTACACACCAACATGCTGGCTTGAATAAAAG 1400
CACCAACTGAAAAAA

```

**b**

MRGATRVSI <del>M</del> LLLV <del>T</del> VSD <del>O</del> AVITGACER <del>I</del> U <del>V</del> CC <del>A</del> CT <del>C</del> AI <del>S</del> U <del>L</del> R <del>G</del> LR <del>M</del> O	50	EG-VEGF
MLL <del>L</del> LL <del>L</del> LP <del>P</del> LLPRAGDA <del>A</del> VI <del>T</del> GAC <del>D</del> <del>I</del> <del>S</del> OC <del>G</del> GG <del>M</del> CAV <del>S</del> U <del>M</del> K <del>S</del> IR <del>I</del> O	48	Bv8 hom
AVITGACER <del>I</del> U <del>C</del> GK <del>G</del> T <del>C</del> CAV <del>S</del> L <del>W</del> K <del>S</del> VR <del>V</del> O	31	VPRA

TP <del>I</del> GREGE <del>E</del> CHP <del>Q</del> SHK <del>V</del> PFF <del>R</del> KRK <del>H</del> H <del>T</del> CP <del>C</del> PL <del>P</del> N <del>L</del> LSR <del>F</del> ED <del>G</del> RYR <del>S</del> MDL	100	EG-VEGF
TP <del>M</del> KL <del>C</del> D <del>S</del> CH <del>P</del> L <del>T</del> R <del>V</del> P <del>F</del> G <del>R</del> R <del>M</del> H <del>H</del> T <del>C</del> PL <del>C</del> LG <del>A</del> LR <del>T</del> S <del>F</del> NR <del>P</del> IC <del>I</del> LAQK	95	Bv8 hom
TP <del>V</del> G <del>T</del> SGE <del>E</del> CHP <del>A</del> SH <del>K</del> TP <del>F</del> S <del>G</del> ORM <del>M</del> H <del>T</del> CP <del>O</del> PN <del>L</del> AC <del>V</del> TP <del>K</del> -R <del>T</del> C <del>L</del> SK	78	VPRA

KNINF 105 EG-VEGF

**c**

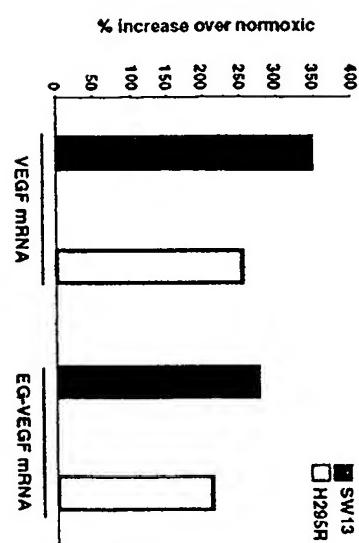
ERDV <del>G</del> AGG <del>T</del> CC <del>A</del> ISL <del>M</del> R <del>G</del> L <del>R</del> -M <del>C</del> T <del>P</del> L <del>G</del> REGEEO <del>O</del> -H <del>F</del> PSH <del>K</del> V <del>P</del> FF <del>R</del> 70 EG-VEGF <del>O</del> DNQRD <del>C</del> OPGL <del>C</del> CAF <del>Q</del> -R <del>G</del> IL <del>F</del> P <del>V</del> C <del>T</del> P <del>L</del> P <del>V</del> EG <del>E</del> IC <del>H</del> PSR <del>L</del> LL <del>D</del> LI	252	hdkk3
DLRSTD <del>C</del> APGL <del>C</del> CA <del>Q</del> -R <del>H</del> FW <del>S</del> K <del>I</del> CK <del>E</del> V <del>L</del> DEC <del>O</del> WT <del>K</del> RR <del>K</del> GS <del>---</del> 215 xdkk1	55	ool
ULNSAD <del>O</del> CKSN <del>C</del> COHD <del>T</del> ISLS <del>N</del> -C <del>A</del> L <del>K</del> AREN <del>S</del> E <del>C</del> SAFTLYG <del>G</del> 105 EG-VEGF		

RKRKH <del>---</del> H <del>T</del> CP <del>C</del> LP <del>N</del> L <del>G</del> SR <del>---</del> -F <del>P</del> D <del>G</del> RYR <del>S</del> MDL <del>K</del> NINF <del>---</del> 290 hdkk3 <del>T</del> WELEPD <del>G</del> ALDR <del>C</del> POASGL <del>L</del> CO <del>---</del> -H <del>S</del> H <del>S</del> L <del>V</del> V <del>V</del> CK <del>P</del> TFVG <del>---</del> <del>H</del> GLE <del>---</del> I <del>F</del> Q <del>R</del> CH <del>O</del> GAGL <del>S</del> ORL <del>Q</del> K <del>G</del> EF <del>T</del> T <del>V</del> PK <del>T</del> S <del>R</del> L <del>H</del> TO <del>Q</del> R <del>H</del> <del>---</del> 254 xdkk1 <del>V</del> Y <del>Y</del> K <del>C</del> P <del>O</del> ER <del>G</del> L <del>I</del> EG <del>D</del> K <del>S</del> LV <del>-</del> GS <del>I</del> T <del>N</del> IN <del>F</del> G <del>I</del> HD <del>V</del> GR <del>S</del> SD <del>---</del> 94 col		
---	--	--



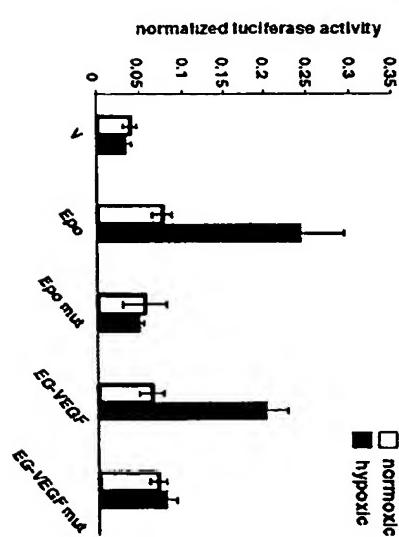
**FIGURE 17 A**

a



**FIGURE 17 B**

b



AGGCCCTACATACGGCTCACACAGCCCTGTTCTGA Epo SEQ ID NO: 16  
 AGGCCCTAATIGCGGGCTCACACAGCCCTGTTCTGA Epo mut SEQ ID NO: 15  
 GCTAAGGGACATGCTATTGATAGGGTCAAGGGAGAT EG-VEGF SEQ ID NO: 17  
 GCTAAGGAATGGTATCATAAGGTGAGGAAGAT EG-VEGF mut SEQ ID NO: 18



FIGURE 18

spleen  
thymus  
prostate  
testis  
ovary  
small intestine  
colon  
PBL  
stomach  
thyroid  
spinal cord  
lymph node  
trachea  
adrenal  
bone marrow  
heart  
brain  
placenta  
lung  
liver  
skeletal muscle  
kidney  
pancreas.  
1.35  
2.4  
kb

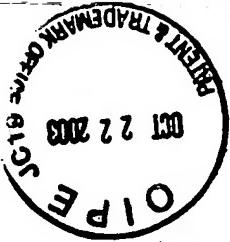
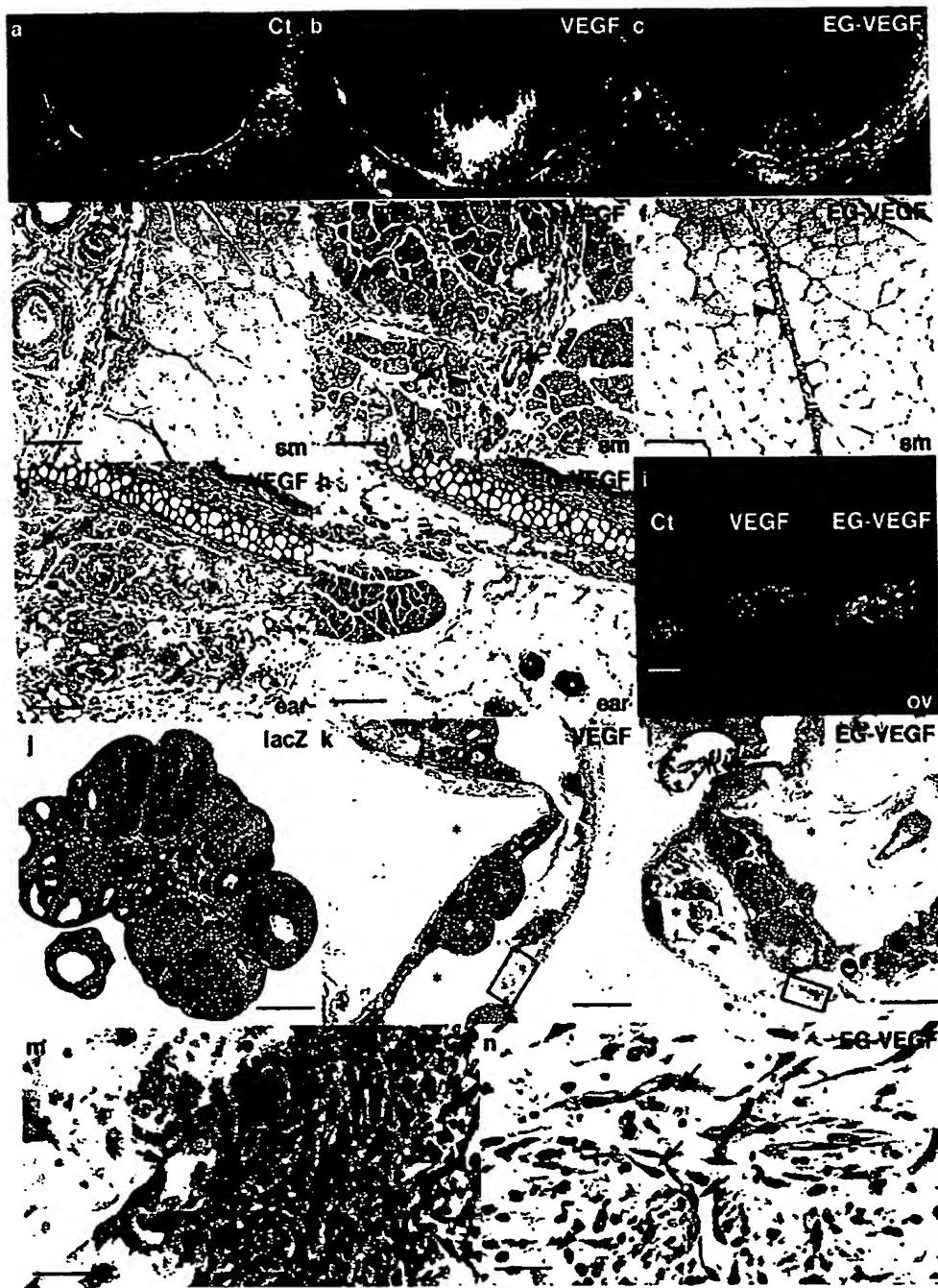


FIGURE 19 A-N





## FIGURE 20 A



## FIGURE 20 B

```

/*
 */
#include <stdio.h>
#include <ctype.h>

#define MAXJMP      16    /* max jumps in a diag */
#define MAXGAP      24    /* don't continue to penalize gaps larger than this */
#define J MPS      1024   /* max jmps in an path */
#define MX         4     /* save if there's at least MX-1 bases since last jmp */

#define DMAT        3     /* value of matching bases */
#define DMIS        0     /* penalty for mismatched bases */
#define DINS0       8     /* penalty for a gap */
#define DINS1       1     /* penalty per base */
#define PINS0       8     /* penalty for a gap */
#define PINS1       4     /* penalty per residue */

struct jmp {
    short          n[MAXJMP];    /* size of jmp (neg for delay) */
    unsigned short x[MAXJMP];    /* base no. of jmp in seq x */
    /* limits seq to 2^16 - 1 */
};

struct diag {
    int            score;        /* score at last jmp */
    long           offset;       /* offset of prev block */
    short          ijmp;         /* current jmp index */
    struct jmp    jp;           /* list of jmps */
};

struct path {
    int            spc;          /* number of leading spaces */
    short          n[J MPS];     /* size of jmp (gap) */
    int            x[J MPS];     /* loc of jmp (last elem before gap) */
};

char            *ofile;        /* output file name */
char            *namex[2];     /* seq names: getseqs() */
char            *prog;         /* prog name for err msgs */
char            *seqx[2];      /* seqs: getseqs() */
int             dmax;         /* best diag: nw() */
int             dmax0;        /* final diag */
int             dna;          /* set if dna: main() */
int             endgaps;      /* set if penalizing end gaps */
int             gapx, gapy;   /* total gaps in seqs */
int             len0, len1;    /* seq lens */
int             ngapx, ngapy; /* total size of gaps */
int             smax;         /* max score: nw() */
int             *xbm;          /* bitmap for matching */
long            offset;        /* current offset in jmp file */
struct diag    *dx;           /* holds diagonals */
struct path    pp[2];        /* holds path for seqs */
char            *calloc(), *malloc(), *index(), *strcpy();
char            *getseq(), *g_malloc();

```



## FIGURE 20 C

```

/* Needleman-Wunsch alignment program
 *
 * usage: progs file1 file2
 * where file1 and file2 are two dna or two protein sequences.
 * The sequences can be in upper- or lower-case and may contain ambiguity
 * Any lines beginning with ';' or '<' are ignored
 * Max file length is 65535 (limited by unsigned short x in the jmp struct)
 * A sequence with 1/3 or more of its elements ACGTU is assumed to be DNA
 * Output is in the file "align.out"
 *
 * The program may create a tmp file in /tmp to hold info about traceback.
 * Original version developed under BSD 4.3 on a vax 8650
 */
#include "nw.h"
#include "day.h"

static _dbval[26] = {
    1,14,2,13,0,0,4,11,0,0,12,0,3,15,0,0,0,5,6,8,8,7,9,0,10,0
};

static _pbval[26] = {
    1, 2|(1<<('D'-'A'))|(1<<('N'-'A')), 4, 8, 16, 32, 64,
    128, 256, 0xFFFFFFFF, 1<<10, 1<<11, 1<<12, 1<<13, 1<<14,
    1<<15, 1<<16, 1<<17, 1<<18, 1<<19, 1<<20, 1<<21, 1<<22,
    1<<23, 1<<24, 1<<25|(1<<('E'-'A'))|(1<<('Q'-'A'))
};

main(ac, av)
{
    int ac;
    char *av[];
{
    prog = av[0];
    if (ac != 3) {
        sprintf(stderr, "usage: %s file1 file2\n", prog);
        sprintf(stderr, "where file1 and file2 are two dna or two protein sequences.\n");
        sprintf(stderr, "The sequences can be in upper- or lower-case\n");
        sprintf(stderr, "Any lines beginning with ';' or '<' are ignored\n");
        sprintf(stderr, "Output is in the file \"align.out\"\n");
        exit(1);
    }
    namex[0] = av[1];
    namex[1] = av[2];
    seqx[0] = getseq(namex[0], &len0);
    seqx[1] = getseq(namex[1], &len1);
    xbm = (dna)? _dbval : _pbval;

    endgaps = 0; /* 1 to penalize endgaps */
    ofile = "align.out"; /* output file */

    nw(); /* fill in the matrix, get the possible jmps */
    readjmps(); /* get the actual jmps */
    print(); /* print stats, alignment */

    cleanup(0); /* unlink any tmp files */
}

```

main

Page 1 of nw.c



## FIGURE 20 D

```

/* do the alignment, return best score: main()
 * dna: values in Fitch and Smith, PNAS, 80, 1382-1386, 1983
 * pro: PAM 250 values
 * When scores are equal, we prefer mismatches to any gap, prefer
 * a new gap to extending an ongoing gap, and prefer a gap in seqx
 * to a gap in seq y.
 */
nw()
{
    char      *px, *py;          /* seqs and ptrs */
    int       *ndely, *dely;      /* keep track of dely */
    int       ndclx, delx;        /* keep track of delx */
    int       *tmp;              /* for swapping row0, row1 */
    int       mis;               /* score for each type */
    int       ins0, ins1;         /* insertion penalties */
    register  id;               /* diagonal index */
    register  jj;               /* jmp index */
    register  *col0, *col1;      /* score for curr, last row */
    register  xx, yy;           /* index into seqs */

    dx = (struct diag *)g_malloc("to get diags", len0+len1+1, sizeof(struct diag));

    ndely = (int *)g_malloc("to get ndely", len1+1, sizeof(int));
    dely = (int *)g_malloc("to get dely", len1+1, sizeof(int));
    col0 = (int *)g_malloc("to get col0", len1+1, sizeof(int));
    col1 = (int *)g_malloc("to get col1", len1+1, sizeof(int));
    ins0 = (dna)? DINS0 : PINS0;
    ins1 = (dna)? DINS1 : PINS1;

    smax = -10000;
    if (endgaps) {
        for (col0[0] = dely[0] = -ins0, yy = 1; yy <= len1; yy++) {
            col0[yy] = dely[yy] = col0[yy-1] - ins1;
            ndely[yy] = yy;
        }
        col0[0] = 0;           /* Waterman Bull Math Biol 84 */
    }
    else
        for (yy = 1; yy <= len1; yy++)
            dely[yy] = -ins0;

    /* fill in match matrix
     */
    for (px = seqx[0], xx = 1; xx <= len0; px++, xx++) {
        /* initialize first entry in col
         */
        if (endgaps) {
            if (xx == 1)
                col1[0] = delx = -(ins0+ins1);
            else
                col1[0] = delx = col0[0] - ins1;
            ndelx = xx;
        }
        else {
            col1[0] = 0;
            delx = -ins0;
            ndelx = 0;
        }
    }
}

```



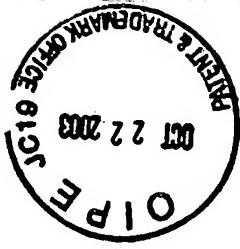
## FIGURE 20 E

```
for (py = seqx[1], yy = 1; yy <= lenl; py++, yy++) {
    mis = col0[yy-1];
    if (dna)
        mis += (xbm[*px-'A']&xbm[*py-'A'])? DMAT : DMIS;
    else
        mis += _day[*px-'A'][*py-'A'];

    /* update penalty for del in x seq;
     * favor new del over ongoing del
     * ignore MAXGAP if weighting endgaps
     */
    if (endgaps || ndely[yy] < MAXGAP) {
        if (col0[yy] - ins0 >= dely[yy]) {
            dely[yy] = col0[yy] - (ins0+ins1);
            ndely[yy] = 1;
        } else {
            dely[yy] = ins1;
            ndely[yy]++;
        }
    } else {
        if (col0[yy] - (ins0+ins1) >= dely[yy]) {
            dely[yy] = col0[yy] - (ins0+ins1);
            ndely[yy] = 1;
        } else
            ndely[yy]++;
    }

    /* update penalty for del in y seq;
     * favor new del over ongoing del
     */
    if (endgaps || ndelx < MAXGAP) {
        if (col1[yy-1] - ins0 >= delx) {
            delx = col1[yy-1] - (ins0+ins1);
            ndelx = 1;
        } else {
            delx = ins1;
            ndelx++;
        }
    } else {
        if (col1[yy-1] - (ins0+ins1) >= delx) {
            delx = col1[yy-1] - (ins0+ins1);
            ndelx = 1;
        } else
            ndelx++;
    }

    /* pick the maximum score; we're favoring
     * mis over any del and delx over dely
     */
}
```



## FIGURE 20 F

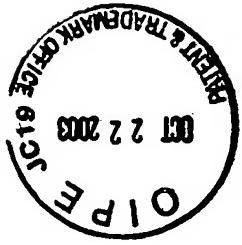
...nw

```

id = xx - yy + len1 - 1;
if (mis >= delx && mis >= dely[yy])
    col1[yy] = mis;
else if (delx >= dely[yy]) {
    col1[yy] = delx;
    ij = dx[id].ijmp;
    if (dx[id].jp.n[0] && (!dn1 || (ndelx >= MAXJMP
        && xx > dx[id].jp.x[ij]+MX) || mis > dx[id].score+DINS0)) {
        dx[id].ijmp++;
        if (++ij >= MAXJMP) {
            writejmps(id);
            ij = dx[id].ijmp = 0;
            dx[id].offset = offset;
            offset += sizeof(struct jmp) + sizeof(offset);
        }
    }
    dx[id].jp.n[ij] = ndelx;
    dx[id].jp.x[ij] = xx;
    dx[id].score = delx;
}
else {
    col1[yy] = dely[yy];
    ij = dx[id].ijmp;

if (dx[id].jp.n[0] && (!dn1 || (ndely[yy] >= MAXJMP
    && xx > dx[id].jp.x[ij]+MX) || mis > dx[id].score+DINS0)) {
    dx[id].ijmp++;
    if (++ij >= MAXJMP) {
        writejmps(id);
        ij = dx[id].ijmp = 0;
        dx[id].offset = offset;
        offset += sizeof(struct jmp) + sizeof(offset);
    }
}
dx[id].jp.n[ij] = -ndely[yy];
dx[id].jp.x[ij] = xx;
dx[id].score = dely[yy];
}
if (xx == len0 && yy < len1) {
    /* last col
    */
    if (endgaps)
        col1[yy] = ins0+ins1*(len1-yy);
    if (col1[yy] > smax) {
        smax = col1[yy];
        dmax = id;
    }
}
}
if (endgaps && xx < len0)
    col1[yy-1] = ins0+ins1*(len0-xx);
if (col1[yy-1] > smax) {
    smax = col1[yy-1];
    dmax = id;
}
tmp = col0; col0 = col1; col1 = tmp;
}

(void) free((char *)ndely);
(void) free((char *)dely);
(void) free((char *)col0);
(void) free((char *)col1);}
```



## FIGURE 20 G

```

/*
 *
 * print() -- only routine visible outside this module
 *
 * static:
 * getmat() -- trace back best path, count matches: print()
 * pr_align() -- print alignment of described in array p[]: print()
 * dumpblock() -- dump a block of lines with numbers, stars: pr_align()
 * nums() -- put out a number line: dumpblock()
 * putline() -- put out a line (name, [num], seq, [num]): dumpblock()
 * stars() - put a line of stars: dumpblock()
 * stripname() -- strip any path and prefix from a seqname
 */

#include "nw.h"

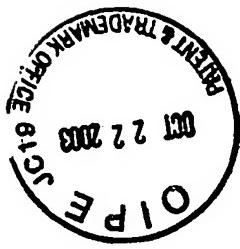
#define SPC      3
#define P_LINE   256      /* maximum output line */
#define P_SPC    3         /* space between name or num and seq */

extern _day[26][26];
int olen;           /* set output line length */
FILE *fx;           /* output file */

print()
{
    int     lx, ly, firstgap, lastgap;      /* overlap */

    if ((fx = fopen(ofile, "w")) == 0) {
        fprintf(stderr, "%s: can't write %s\n", prog, ofile);
        cleanup(1);
    }
    sprintf(fx, "<first sequence: %s (length = %d)\n", namex[0], len0);
    sprintf(fx, "<second sequence: %s (length = %d)\n", namex[1], len1);
    olen = 60;
    lx = len0;
    ly = len1;
    firstgap = lastgap = 0;
    if (dmax < len1 - 1) {          /* leading gap in x */
        pp[0].spc = firstgap = len1 - dmax - 1;
        ly -= pp[0].spc;
    }
    else if (dmax > len1 - 1) {    /* leading gap in y */
        pp[1].spc = firstgap = dmax - (len1 - 1);
        lx -= pp[1].spc;
    }
    if (dmax0 < len0 - 1) {        /* trailing gap in x */
        lastgap = len0 - dmax0 - 1;
        lx -= lastgap;
    }
    else if (dmax0 > len0 - 1) {  /* trailing gap in y */
        lastgap = dmax0 - (len0 - 1);
        ly -= lastgap;
    }
    getmat(lx, ly, firstgap, lastgap);
    pr_align();
}

```



## FIGURE 20 H

```

/*
 * trace back the best path, count matches
 */
static
getmat(lx, ly, firstgap, lastgap)
    int      lx, ly;          /* "core" (minus endgaps) */
    int      firstgap, lastgap; /* leading/trailing overlap */
{
    int      nm, i0, i1, siz0, siz1;
    char     outx[32];
    double   pct;
    register int n0, n1;
    register char *p0, *p1;

    /* get total matches, score
     */
    i0 = i1 = siz0 = siz1 = 0;
    p0 = seqx[0] + pp[1].spc;
    p1 = seqx[1] + pp[0].spc;
    n0 = pp[1].spc + 1;
    n1 = pp[0].spc + 1;

    nm = 0;
    while (*p0 && *p1) {
        if (siz0) {
            p1++;
            n1++;
            siz0--;
        }
        else if (siz1) {
            p0++;
            n0++;
            siz1--;
        }
        else {
            if (xbm[*p0-'A']&xbm[*p1-'A'])
                nm++;
            if (n0++ == pp[0].n[i0])
                siz0 = pp[0].n[i0++];
            if (n1++ == pp[1].n[i1])
                siz1 = pp[1].n[i1++];
            p0++;
            p1++;
        }
    }
    /* pct homology:
     * if penalizing endgaps, base is the shorter seq
     * else, knock off overhangs and take shorter core
     */
    if (endgaps)
        lx = (len0 < len1)? len0 : len1;
    else
        lx = (lx < ly)? lx : ly;
    pct = 100.*double(nm)/(double)lx;
    sprintf(fx, "\n");
    sprintf(fx, "<%d match%s in an overlap of %d: %.2f percent similarity\n",
            nm, (nm == 1)? ":" : "es", lx, pct);
}

```



## FIGURE 20 I

```

fprintf(fx, "<gaps in first sequence: %d", gapx);           ...getmat
if (gapx) {
    (void) sprintf(outx, " (%d %s%s)", ngapx, (dna)? "base": "residue", (ngapx == 1)? ":" : "s");
    sprintf(fx, "%s", outx);

fprintf(fx, ", gaps in second sequence: %d", gapy);
if (gapy) {
    (void) sprintf(outx, " (%d %s%s)", ngapy, (dna)? "base": "residue", (ngapy == 1)? ":" : "s");
    sprintf(fx, "%s", outx);
}
if (dna)
    fprintf(fx,
    "n<score: %d (match = %d, mismatch = %d, gap penalty = %d + %d per base)\n",
    smax, DMAT, DMIS, DINSO, DINSI);
else
    fprintf(fx,
    "n<score: %d (Dayhoff PAM 250 matrix, gap penalty = %d + %d per residue)\n",
    smax, PINS0, PINS1);
if (endgaps)
    fprintf(fx,
    "<endgaps penalized. left endgap: %d %s%s, right endgap: %d %s%s\n",
    firstgap, (dna)? "base" : "residue", (firstgap == 1)? ":" : "s",
    lastgap, (dna)? "base" : "residue", (lastgap == 1)? ":" : "s");
else
    fprintf(fx, "<endgaps not penalized\n");
}

static nm;          /* matches in core -- for checking */
static lmax;        /* lengths of stripped file names */
static ij[2];        /* jmp index for a path */
static nc[2];        /* number at start of current line */
static ni[2];        /* current elem number -- for gapping */
static siz[2];
static char *ps[2];    /* ptr to current element */
static char *po[2];    /* ptr to next output char slot */
static char out[2][P_LINE]; /* output line */
static char star[P_LINE]; /* set by stars() */

/*
 * print alignment of described in struct path pp[]
 */
static
pr_align()
{
    int nm;          /* char count */
    int more;
    register i;

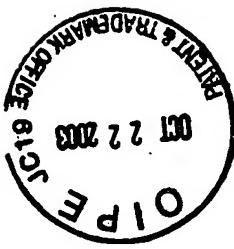
    for (i = 0, lmax = 0; i < 2; i++) {
        nn = stripname(namex[i]);
        if (nn > lmax)
            lmax = nn;

        nc[i] = 1;
        ni[i] = 1;
        siz[i] = ij[i] = 0;
        ps[i] = seqx[i];
        po[i] = out[i];
    }
}

```

pr\_align

Page 3 of nwprint.c



## FIGURE 20 J

```

...pr_align

for (nn = nm = 0, more = 1; more; ) {
    for (i = more = 0; i < 2; i++) {
        /*
         * do we have more of this sequence?
         */
        if (!*ps[i])
            continue;

        more++;

        if (pp[i].spc) { /* leading space */
            *po[i]++ = ' ';
            pp[i].spc--;
        }
        else if (siz[i]) { /* in a gap */
            *po[i]++ = '^';
            siz[i]--;
        }
        else { /* we're putting a seq element
                 */
            *po[i] = *ps[i];
            if (islower(*ps[i]))
                *ps[i] = toupper(*ps[i]);
            po[i]++;
            ps[i]++;

            /*
             * are we at next gap for this seq?
             */
            if (ni[i] == pp[i].x[i][i]) {
                /*
                 * we need to merge all gaps
                 * at this location
                 */
                siz[i] = pp[i].n[i][i]++;
                while (ni[i] == pp[i].x[i][i])
                    siz[i] += pp[i].n[i][i]++;
            }
            ni[i]++;
        }
    }
    if (++nn == olen || !more && nn) {
        dumpblock();
        for (i = 0; i < 2; i++)
            po[i] = out[i];
        nn = 0;
    }
}

/*
 * dump a block of lines, including numbers, stars: pr_align()
 */
static
dumpblock
{
    register i;

    for (i = 0; i < 2; i++)
        *po[i] = '\0';
}

```



## FIGURE 20 K

```

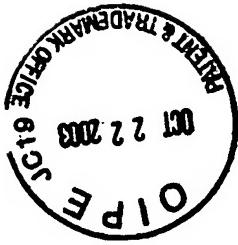
...dumpblock

(void) putc('\n', fx);
for (i = 0; i < 2; i++) {
    if (*out[i] && (*out[i] != ' ' || *(po[i]) != ' ')) {
        if (i == 0)
            nums(i);
        if (i == 0 && *out[1])
            stars();
        putline(i);
        if (i == 0 && *out[1])
            fprintf(fx, star);
        if (i == 1)
            nums(i);
    }
}
/*
 * put out a number line: dumpblock()
 */
static
nums(ix)
{
    int      ix;      /* index in out[] holding seq line */
    char      nline[P_LINE];
    register   i, j;
    register char  *pn, *px, *py;

    for (pn = nline, i = 0; i < lmax+P_SPC; i++, pn++)
        *pn = ' ';
    for (i = nc[ix], py = out[ix]; *py; py++, pn++) {
        if (*py == ' ' || *py == '-')
            *pn = ' ';
        else {
            if (i%10 == 0 || (i == 1 && nc[ix] != 1)) {
                j = (i < 0)? -i : i;
                for (px = pn; j /= 10, px--)
                    *px = j%10 + '0';
                if (i < 0)
                    *px = '-';
            }
            else
                *pn = ' ';
            i++;
        }
    }
    *pn = '\0';
    nc[ix] = i;
    for (pn = nline; *pn; pn++)
        (void) putc(*pn, fx);
    (void) putc('\n', fx);
}

/*
 * put out a line (name, [num], seq, [num]): dumpblock()
 */
static
putline(ix)
{
    int      ix;

```



## FIGURE 20 L

```

...putline

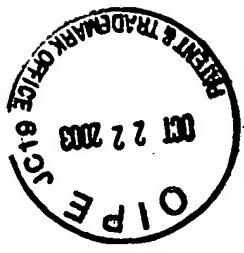
int          i;
register char *px;
for (px = namex[ix], i = 0; *px && *px != ':'; px++, i++)
    (void) putc(*px, fx);
for (; i < lmax+P_SPC; i++)
    (void) putc(' ', fx);

/* these count from 1:
 * ni[] is current element (from 1)
 * nc[] is number at start of current line
 */
for (px = out[ix]; *px; px++)
    (void) putc(*px&0x7F, fx);
(void) putc('\n', fx);
}

/*
 * put a line of stars (seqs always in out[0], out[1]): dumpblock()
 */
stars()                                stars
{
    int          i;
    register char *p0, *p1, cx, *px;

    if (!*out[0] || (*out[0] == ' ' && *(po[0]) == ')') ||
        !*out[1] || (*out[1] == ' ' && *(po[1]) == ')')
        return;
    px = star;
    for (i = lmax+P_SPC; i; i--)
        *px++ = ' ';
    for (p0 = out[0], p1 = out[1]; *p0 && *p1; p0++, p1++) {
        if (isalpha(*p0) && isalpha(*p1)) {
            if (xbm[*p0-'A']&xbm[*p1-'A']) {
                cx = "*";
                nm++;
            }
            else if (!dma && _day[*p0-'A'][*p1-'A'] > 0)
                cx = ".";
            else
                cx = ",";
        }
        else
            cx = " ";
        *px++ = cx;
    }
    *px++ = "\n";
    *px = '\0';
}

```



## FIGURE 20 M

```
/*
 * strip path or prefix from pn, return len: pr_align()
 */
static
stripname(pn)
    char    *pn;    /* file name (may be path) */
{
    register char    *px, *py;

    py = 0;
    for (px = pn; *px; px++)
        if (*px == '/')
            py = px + 1;
    if (py)
        (void) strepy(pn, py);
    return(strlen(pn));
}
```

Page 7 of nwprint.c



## FIGURE 20 N

```

/*
 * cleanup() -- cleanup any tmp file
 * getseq() -- read in seq, set dna, len, maxlen
 * g_calloc() -- calloc() with error checkin
 * readjmps() -- get the good jmps, from tmp file if necessary
 * writejmps() -- write a filled array of jmps to a tmp file: nw0
 */
#include "nw.h"
#include <sys/file.h>

char *jname = "/tmp/homgXXXXXX";           /* tmp file for jmps */
FILE *fj;

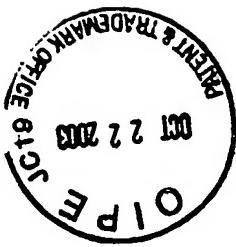
int cleanup();                                /* cleanup tmp file */

/* remove any tmp file if we blow
 */
cleanup(i)
    int i;
{
    if (fj)
        (void) unlink(jname);
    exit(i);
}

/*
 * read, return ptr to seq, set dna, len, maxlen
 * skip lines starting with ';', '<', or '>'
 * seq in upper or lower case
 */
char *
getseq(file, len)
    char *file;   /* file name */
    int len;     /* seq len */
{
    char line[1024], *pseq;
    register char *px, *py;
    int natgc, tlen;
    FILE *fp;

    if ((fp = fopen(file, "r")) == 0) {
        fprintf(stderr, "%s: can't read %s\n", prog, file);
        exit(1);
    }
    tlen = natgc = 0;
    while (fgets(line, 1024, fp)) {
        if (*line == ';' || *line == '<' || *line == '>')
            continue;
        for (px = line; *px != '\n'; px++)
            if (isupper(*px) || islower(*px))
                tlen++;
    }
    if ((pseq = malloc((unsigned)(tlen+6))) == 0) {
        fprintf(stderr, "%s: malloc() failed to get %d bytes for %s\n", prog, tlen+6, file);
        exit(1);
    }
    pseq[0] = pseq[1] = pseq[2] = pseq[3] = '\0';
}

```



## FIGURE 20 O

```

...getseq

py = pseq + 4;
*tlen = tlen;
rewind(fp);

while (fgets(line, 1024, fp)) {
    if (*line == ';' || *line == '<' || *line == '>')
        continue;
    for (px = line; *px != '\n'; px++) {
        if (isupper(*px))
            *py++ = *px;
        else if (islower(*px))
            *py++ = toupper(*px);
        if (index("ATGCU", *(py-1)))
            natgc++;
    }
    *py++ = '\0';
    *py = '\0';
    (void) fclose(fp);
    dna = natgc > (tlen/3);
    return(pseq+4);
}

char *
g_calloc(msg, nx, sz)
    char    *msg;           /* program, calling routine */
    int     nx, sz;         /* number and size of elements */
{
    char    *px, *calloc();

    if ((px = calloc((unsigned)nx, (unsigned)sz)) == 0) {
        if (*msg) {
            fprintf(stderr, "%s: g_calloc() failed %s (n=%d, sz=%d)\n", prog, msg, nx, sz);
            exit(1);
        }
    }
    return(px);
}

/*
 * get final jmps from dx[] or tmp file, set pp[], reset dmax: main()
 */
readjmps()                                         readjmps
{
    int             fd = -1;
    int             siz, i0, i1;
    register int i, j, xx;

    if (fj) {
        (void) fclose(fj);
        if ((fd = open(jname, O_RDONLY, 0)) < 0) {
            fprintf(stderr, "%s: can't open() %s\n", prog, jname);
            cleanup(1);
        }
    }
    for (i = i0 = i1 = 0, dmax0 = dmax, xx = len0; i++) {
        while (1) {
            for (j = dx[dmax].jmp; j >= 0 && dx[dmax].jp.x[j] >= xx; j--)

```



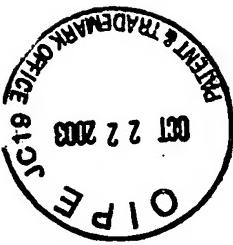
## FIGURE 20 P

```

...readjmps

if (j < 0 && dx[dmax].offset && jj) {
    (void) lseek(fd, dx[dmax].offset, 0);
    (void) read(fd, (char *)&dx[dmax].jp, sizeof(struct jmp));
    (void) read(fd, (char *)&dx[dmax].offset, sizeof(dx[dmax].offset));
    dx[dmax].ijmp = MAXJMP-1;
}
else
    break;
}
if (i >= JMPS) {
    fprintf(stderr, "%s: too many gaps in alignment\n", prog);
    cleanup(1);
}
if (j >= 0) {
    siz = dx[dmax].jp.n[j];
    xx = dx[dmax].jp.x[j];
    dmax += siz;
    if (siz < 0) { /* gap in second seq */
        pp[1].n[i1] = -siz;
        xx += siz;
        /* id = xx - yy + len1 - 1
        */
        pp[1].x[i1] = xx - dmax + len1 - 1;
        gappy++;
        ngappy -= siz;
    /* ignore MAXGAP when doing endgaps */
        siz = (-siz < MAXGAP || endgaps)? -siz : MAXGAP;
        i1++;
    }
    else if (siz > 0) { /* gap in first seq */
        pp[0].n[i0] = siz;
        pp[0].x[i0] = xx;
        gapx++;
        ngapx += siz;
    /* ignore MAXGAP when doing endgaps */
        siz = (siz < MAXGAP || endgaps)? siz : MAXGAP;
        i0++;
    }
}
else
    break;
}
/* reverse the order of jmps
 */
for (j = 0, i0--; j < i0; j++) {
    i = pp[0].x[j]; pp[0].n[j] = pp[0].n[i0]; pp[0].n[i0] = i;
    i = pp[0].x[j]; pp[0].x[j] = pp[0].x[i0]; pp[0].x[i0] = i;
}
for (j = 0, i1--; j < i1; j++) {
    i = pp[1].n[j]; pp[1].n[j] = pp[1].n[i1]; pp[1].n[i1] = i;
    i = pp[1].x[j]; pp[1].x[j] = pp[1].x[i1]; pp[1].x[i1] = i;
}
if (fd >= 0)
    (void) close(fd);
if (jj) {
    (void) unlink(jname);
    jj = 0;
    offset = 0;
}
}
}

```



## FIGURE 20 Q

```
/*
 * write a filled jmp struct offset of the prev one (if any): nw()
 */
writejmps(ix)
    int      ix;
{
    char    *mktemp();

    if (!fj) {
        if (mktemp(jname) < 0) {
            fprintf(stderr, "%s: can't mktemp() %s\n", prog, jname);
            cleanup(1);
        }
        if ((fj = fopen(jname, "w")) == 0) {
            fprintf(stderr, "%s: can't write %s\n", prog, jname);
            exit(1);
        }
    }
    (void) fwrite((char *)&dx[ix].jp, sizeof(struct jmp), 1, fj);
    (void) fwrite((char *)&dx[ix].offset, sizeof(dx[ix].offset), 1, fj);
}
```

Page 4 of nwsnbr.c

W:\DOCS\GRD\FIGURES.GENENT1516A